Program 6 Hints

There are basically two different things that need to be done in this application. (1) You need to read in the words to two different dictionaries, one with about 30000 regular words that might appear in a text message (let's call this one `dictionary`), and another with about 100 illegal words that might also appear in a text message but if included would cause the message to be censored by your application (let's call this one `illegal`). (2) You need to process a set of text messages to decide whether to send or not send the message based on the specified protocol.

PART 1

Each word in either dictionary has a limit of 29 characters (so the array must hold 30 characters to have room for the \n). So we would have the following (you should declare constants for these various array sizes):

```
char dictionary[30000][30];

char illegal[100][30];
```

To tackle the first problem, you must be aware of the input file structure for the input file that is to be processed. The format of this file is:

first line: an integer value indicating the number of words in the dictionary – let's call this value, `numWordsInDictionary`.

`numWordsInDictionary` lines: each line contains 1 word in the dictionary.

next line: an integer value indicating the number of words in the illegal word dictionary. – let's call this value `numBadWords`.

`numBadWords` lines: each line contains 1 illegal word.

next line: an integer value indicating the number of text messages that are to be processed. Let's call this value, `numMessages`.

next `2*numMessages` lines: first line is a timestamp, second line starts with an integer value indicating the number of words in the message followed by that number of words.

A single function can be used to read both dictionaries and all messages because they all have the same format, an integer followed by that number of strings. Let's call this function `readFile`. This function will require two parameters, the file pointer and the array of strings it is filling up. It can simply return the integer value it reads to the caller.

In your `main` function, you can initialize the file pointer to point to the file named `textmsg.txt` and simply pass the file pointer and the proper array of strings to the `readFile` function. Thus, once the file is opened and the file pointer is assigned you need to do the following:

```
numWordsInDictionary = readFile(filepointer, dictionary);

numBadWords = readFile(filepointer, illegal);
```

To test this part of your program, simply print out the words in both dictionaries, one word/line. If you can read and the print the dictionaries properly, then both dictionaries are now filled and you're ready to begin processing text messages, or handling the second part of the assignment.

PART 2

The number of messages to be processed and the message timestamps do not follow the same format as the other parts of the file, so I would suggest just using normal `fscanf` statements to handle reading those items. However, each message follows the same format as the dictionaries, so I would reuse the same `readFile` function again to read each message.

IMPORTANT NOTE: A message is not a simple string of characters. A message is an array of strings, just like a dictionary in that sense. Each string consists of the number of words given by the integer in the first position of the line. So the format of a message is:

```
char textMessage[50][30];
```

Once again, you can use the `readFile` function to read each text message from the file, because the format is the same: an integer (the number of words in the message), followed by that many strings (words) which comprise the message.

When you're first developing your program, I would simply skip over the timestamp. You need to read it of course, but just ignore it for now, you can go back and deal with it once you know that you can process the messages based on text alone. I'll proceed with this assumption.

To check the validity of a message (independent of the timestamp) you need to do three things: (1) check for misspelled words – 3 or more will cause the censoring of the message, (2) check for any illegal words in the message – 1 will cause the message to be censored, and (3) if the phrase "I Love You" appears in the message, it will be censored. It really doesn't matter too much in what order you do things, but from an ease of work point of view, I looked for illegal words first, since there are not too many of them and only 1 of them included in the message is enough to cause the message to be censored. Then I handled misspelled words, and finally checked for the "I Love You" phrase.

For the first two cases, again, a single function can handle both problems, since it is just using a different dictionary on the two calls. The function needs to check every word (string) in the

message against the words (strings) in the dictionary passed to it. I called this function `wordCount` and it returns an integer, which is simply the number of words it found in the dictionary that matched words in the message. So for example, the call:

```
numBadWords = wordCount(message, messageLength, illegal, numBadWords);
```

will return the number of words in the message passed to it that were matched in the illegal word dictionary. If this number is > 0, the message included an illegal word and must be censored, otherwise, the message contained no illegal words.

A similar call using the regular word dictionary will return the number of words in the message that were found in the dictionary. Suppose a message contained 30 words and the function returned a value of 26. This would mean that 26 of the words in the message were in the dictionary and 4 were not. Thus, 4 words were misspelled and the message will need to be censored (note, that we've already passed it for containing no illegal words – if there had been illegal words in the message we wouldn't even be doing this step!).

IMPORTANT NOTE: The text messages contain a mix of lower and upper case letters. However, dictionary entries are all lower case and the censoring protocol is not case sensitive, so the first thing that the `wordCount` function does is use the `strcpy` function to copy the current word in the message into a temporary string and then this is passed to a function that uses the `toLower` function (see Strings In C – Part 2 page 19 for `strcpy` and Strings In C – Part 4 – page 3 for `toLower` – note that you need to include `ctype.h` header to use the `toLower` function) to convert the word to all lower case symbols.

Finally, if the message as passed through both of the steps above, our final test is to see if it contains the phrase "I Love You". I wrote a separate function to handle this case (it too first converts each word to lower case). Basically this function will make use of the `strcmp` function (see Strings In C – Part 2, page 19 and Strings In C – Part 3, pages 5-9). What happens in this function is that each word in the message is compared (using `strcmp`) with "`i`", if a match is found (strcmp returns 0), then a counter is incremented to indicate that we've seen the first word in the phrase, if the next word in the message matches "`love`", the counter is again incremented, and if the next word matches "`you`", we've found all three words together and the message will be censored. If at any point during the incrementing the next word in the message does not match the word in the phrase, the counters are reset and we continue on.

Assuming the message is valid (its passed all these tests), then the message is simply printed out as being sent.

At this point, you've successfully determined, based on message content, whether a message is to be censored or passed on to the recipient.

The Timestamp Component of Part 2

Messages with a timestamp between 7:00am and 12:59pm (the next day) will not be subject to censoring and are simply printed out as being sent to the recipient.

The only messages that must be censored are those messages with a timestamp between 1:00am and 6:59am.

The timestamp line in the input file should be read using a normal fscanf statement, such as:

```
fscanf(filepointer, "%s%s", timestamp, ampm);
```

To handle timestamps, I again wrote a separate function to handle them. This function returns an integer (1 == TRUE, 0 == FALSE) where TRUE is returned if the time is between 7:00am and 12:59pm, and FALSE otherwise.

This is accomplished with 3 `if` statements as follows: (1) using `strcmp` matching `ampm` with "PM", if `strcmp` returns 0, my function returns TRUE – justification: any timestamp with PM will not be censored. (2) `if timestamp[1] != ':'`, my function returns TRUE – justification: if the second location in the timestamp isn't a colon then we must be dealing with a time that is either 10, 11, or 12; all of which are outside the censoring window. (3) `if timestamp[0] > '6'`, my function returns TRUE – justification: the hour digit is a single digit and it must be either 7, 8, or 9; all of which are outside the censoring window. Everything else must be in the time period where censoring will occur, so my function returns FALSE.

I've put another small text message input file on our assignment page – the same place you went to get this document. Later in the week I'll add a few more messages to this file and possibly make another one for you to test with (depends if I get the time though!).

See the next page for description and output of this current file.

The file contains:

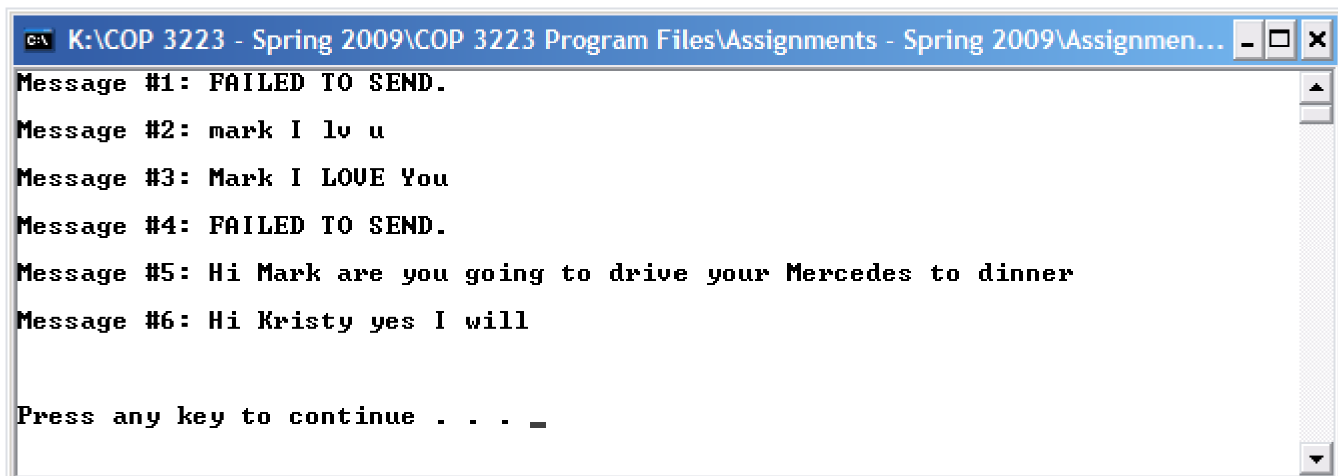
```
43
a
abba
am
are
bye
can
car
day
debi
dinner
do
drive
eat
first
for
fun
going
good
great
hello
hi
i
kiss
kristy
```

```
love
mark
me
mercedes
movie
new
no
not
please
quit
read
stop
time
to
watch
will
yes
you
your
4
dump
hate
jerk
stupid
6
```

mytextmsg - Notepad

```
6
2:30 AM
4 Mark I hate you
2:31 AM
4 mark I lv u
8:03 PM
4 Mark I LOVE You
1:45 AM
6 I aM going to DUMP you
3:30 PM
11 Hi Mark are you going to drive your
Mercedes to dinner
3:31 PM
5 Hi Kristy yes I will
3:32 PM
5 I will watch for you
```

The output:



```
Message #1: FAILED TO SEND.

Message #2: mark I lv u

Message #3: Mark I LOVE You

Message #4: FAILED TO SEND.

Message #5: Hi Mark are you going to drive your Mercedes to dinner

Message #6: Hi Kristy yes I will


Press any key to continue . . .
```